Name: Sahib Bajwa

ID: 107553086

CSCI 3104, Algorithms **Profs. Chen & Grochow**

Problem Set 6 – Due Fri Feb 28 11:55pm **Spring 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Informal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solutions:

- All submissions must be easily legible.
- You should submit your work through the **class Canvas page** only.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please allot at least as many pages per problem (or subproblem) as are allotted in this template.
- For drawing graphs, you may include scans of hand-drawn graphs into your PDF file. **However, the rest of your solution (including the explanation of the graph) must be typed. If your words are not typed, you will get a 0 for that part of the question.**

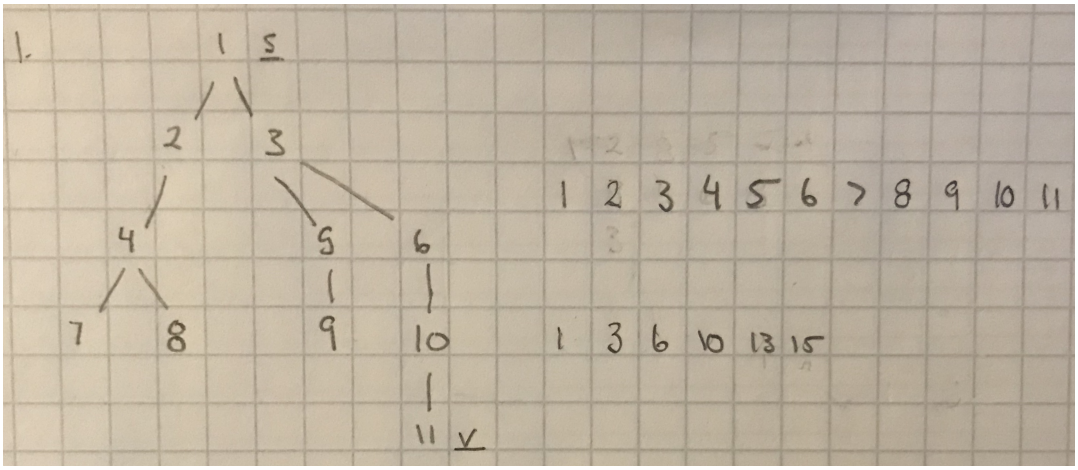
Quicklinks: 1 2 3 4a 4b 4c

Name: Sahib Bajwa
ID: 107553086

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 6 – Due Fri Feb 28 11:55pm **Spring 2020, CU-Boulder**

1. Give an example of a (simple, undirected) graph $G = (V, E)$, a start vertex $s \in V$ and a set of tree edges $E_T \subseteq E$ such that for each vertex $v \in V$, the unique path in the graph (V, E_T) from s to v is a shortest path in G, yet the set of edges E_T cannot be produced by running a breadth-first search on G, no matter how the vertices are ordered. Include an explanation of why your example satisfies the requirements.

   Start vertex s = 1, End vertex v = 11
   The breadth-first search needs to check all vertexes from each branch that appear before 11, which includes a branch that does not include 11. Since breadth-first search checks all other nodes before getting to 11, it does not find the shortest path from s to v in G. The path the breadth-first search creates is [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. The shortest path would have been [1, 3, 6, 10, 11].

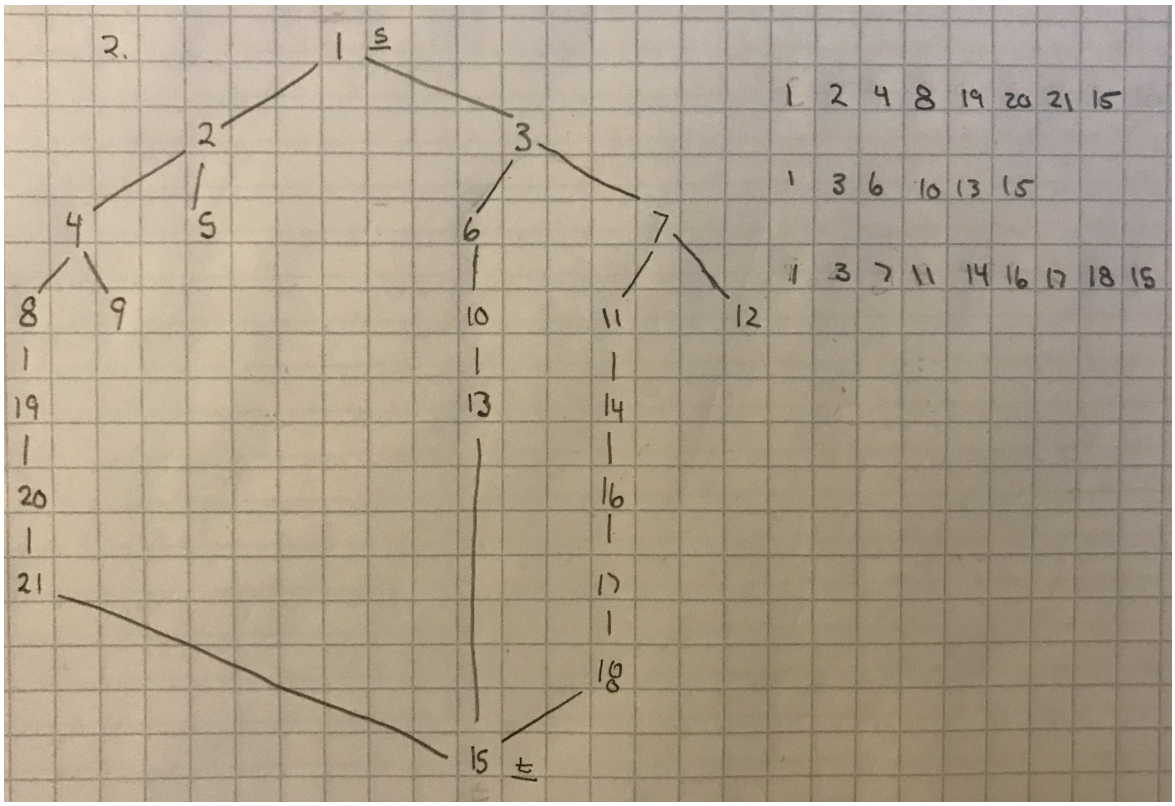


Name: Sahib Bajwa
ID: 107553086

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 6 – Due Fri Feb 28 11:55pm **Spring 2020, CU-Boulder**

2. A *simple $s \to t$ path* in a graph G is a path in G starting at s, ending at t, and never visiting the same vertex twice. Give an example graph (simple, undirected, unweighted) $G = (V, E)$ and vertices $s, t \in V$ such that DFS finds a path from s to t which is neither a shortest path nor a longest simple path. Detail the execution of DFS (list the contents of the queue at each step, and which vertex it pops off the queue), show the final $s \to t$ path it finds, show a shorter $s \to t$ path, and a longer simple $s \to t$ path.

Start vertex s = 1, End vertex t = 15
The DFS goes as far as it can along one path until it can go no further, it then explores the next path. For this graph, DFS takes the path of [1, 2, 4, 8, 19, 20, 21, 15]. The shortest path that you could take would be [1, 3, 6, 10, 13, 15]. The longest path you could take would be [1, 3, 7, 11, 14, 16, 17, 18, 15].

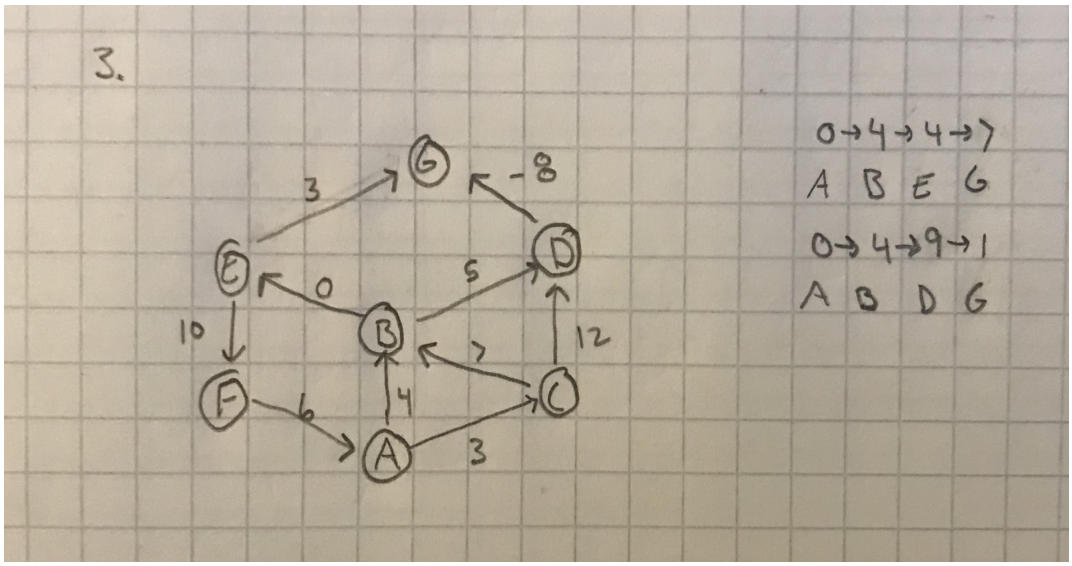


Name: Sahib Bajwa
ID: 107553086

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 6 – Due Fri Feb 28 11:55pm **Spring 2020, CU-Boulder**

3. Give an example of a simple *directed, weighted* graph $G = (V, E, w\colon E \to \mathbb{R})$ and vertices $s, t \in V$ such that Dijkstra's algortihm started at s does *not* find the shortest $s \to t$ path. *Hint:* You will need to use negative edge weights. (Note: this shows that for finding shortest paths, the greedy choice property *fails* in the presence of negative edge weights. Do you see why?)

Start vertex s = A, End vertex t = G
Dijkstra's algorithm will go down the path with the lowest cumulative weight. If there is a negative weight at the end of a path that has a higher cumulative weight than an entire different path, Dijkstra's algorithm will never get to the negative weight. Dijkstra's algorithm gives us the path [A, B, E, G] with a total weight of 7. The shortest path would have been the path [A, B, D, G] with a total weight of 1 since the path from D to G is a negative weight.



Name: Sahib Bajwa
ID: 107553086

4. You have three batteries, with 4200, 2700, and 1600 mAh (milli-Amp-hours), respectively. The 2700 and 1600-mAh batteries are fully charged (containing 2700 mAh and 1600 mAh, respectively), while the 4200-mAh battery is empty, with 0 mAh. You have a battery transfer device which has a "source" battery position and a "target" battery position. When you place two batteries in the device, it instantaneously transfers as many mAh from the source battery to the target battery as possible. Thus, this device stops the transfer either when the source battery has no mAh remaining or when the destination battery is fully charged (whichever comes first).

   But battery transfers aren't free! The battery device is also hooked up to your phone by bluetooth, and automatically charges you a number of cents equal to however many mAh it just transfered.

   The goal in this problem is to determine whether there exists a sequence of transfers that leaves exactly 1200 mAh either in the 2700-mAh battery or the 1600-mAh battery, and if so, how little money you can spend to get this result.

(a) Rephrase this is as a graph problem. Give a precise definition of how to model this problem as a graph, and state the specific question about this graph that must be answered.

You have a graph where at every vortex you have an option of going to one of six different situations. In each situation, you have done a different operation of moving charge from one of the batteries (4200 mAh, 2700 mAh, 1600 mAh) to another. Each time you move to a vortex, the amount of charge the last operation moved is tallied in cents. The batteries start with the 2700 mAh and 1600 mAh batteries full and the 4200 mAh battery empty. Transverse the graph in order to reach a point where one of the 2700 mAh or 1600 mAh batteries has 1200 mAh of battery left. Try and minimize the amount of cents tallied.

(b) What algorithm should you apply to solve this problem?

I think both Dijkstra's algorithm and BFS may work on this problem. Since you are essentially using the weights in a different way though, I think BFS will be the best way to solve this problem as the first time BFS reaches the answer will have been the shortest possible way to reach it, hopefully giving the smallest amount of cents used as well. If we implement the graph correctly though, Dijkstra's algorithm will be the best way to solve this problem.

(c) Apply that algorithm to the question. Report and justify your answer—both the sequence of steps and the total cost. To justify your answer here likely means you will have to detail the steps the algorithm takes.

B1 = 4200 mAh battery, B2 = 2700 mAh battery, B3 = 1600 mAh battery The sequence we should take to solve this problem would be to dump B2 into B1 with a cost of 2700. Then to dump B3 into B1 for a cost of 1500. Then to dump B1 into B2 for a cost of 2700. Then to dump B2 into B3 for a cost of 1500. This leaves 1500 in B1, 1200 in B2, and 1600 in B3 with a total cost of 8400 cents. Dijkstra's algorithm compares paths until and as one outweighs the other (i.e. one path costs more cents than the other), Dijkstra's algorithm follows the path with less weight. Using that and going along the paths, this was the answer I got to.

Here is a visual example of the transfers:

Step	Cost (per step)	Transfer	4200 mAh	2700 mAh	1600 mAh
0	0	-	0	2700	1600
1	2700	B2 $\rightarrow$ B1	2700	0	1600
2	1500	B3 $\rightarrow$ B1	4200	0	100
3	2700	B1 $\rightarrow$ B2	1500	2700	100
4	1500	B2 $\rightarrow$ B3	1500	1200	1600
Totals:	8400	-	1500	1200	1600